Using the Event Coordination Notation for Validation

Kindler, Ekkart; Egilsson, Petur Ingi ; Hillah, Lom Messan

Published in:
Algorithms and Tools for Petri Nets - Proceedings of the Workshop AWPN 2018

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Using the Event Coordination Notation for Validation

Ekkart Kindler[1], Pétur Ingi Egilsson[1], and Lom Messan Hillah[2]

[1] DTU Compute, Technical University of Denmark
[2] Univ. Paris Nanterre & Sorbonne Université CNRS, LIP6, Paris

1 Introduction

The *Event Coordination Notation* (*ECNO*) can be used for defining how related model elements in a system must coordinate their behaviour. It can be used to model software systems and then generate the code from these models fully automatically [1]. In a previous study, we demonstrated that a large software system, a workflow management system, could be modelled by ECNO and then the code for it could be generated automatically [2, 3].

The ECNO, however, can also be used for rapid prototyping, where the generated code is just a means to try out specific behaviour; the generated code is not the final result. Once the prototyping results in the desired behaviour, the system is implemented manually; for example, when the code runs in a distributed way[3]. In that case, the ECNO model can serve as more than just a prototype for the final implementation: in the end, we can use the original ECNO model to validate the final implementation.

This idea first came up when we used ECNO for modelling a large and distributed banking system [4] for prototyping. It roughly took a day or two to come up with a first model and play with the automatically generated code. Initially, we used this model for prototyping only. Later, in his masters' project, Egilsson [5] designed and implemented an extension of the ECNO Tool, which then allowed us to validate an implementation of the banking system against the ECNO prototype.

This paper discusses this idea of validating manual implementations against ECNO models and some of its main ingredients. In Sect. 2, we briefly introduce the core ideas of ECNO by using an example. In Sect. 3, we discuss the state space of ECNO models. In Sect. 4, we describe traces as an abstraction of the observed behaviour of the manually implemented system. In Sect. 5, we show how a trace of the implementation can be validated against the ECNO model by mapping the trace to the state space of the ECNO model.

2 ECNO by Example

To explain the main idea of ECNO, we use the simple introductory example from the ECNO report [1]. It models a company in which *workers* are required

[3] Up to now, the ECNO code generator cannot generate such code automatically.

to jointly do some *jobs*. Only if all the workers that are *needed* for the job are available, the job can be done. To make things slightly more interesting, we assume that the workers share *cars* for coming in for work and for leaving again. Therefore, the workers sharing the same car will *arrive* and *depart* together. And only when a worker is at *work*, the worker is available for *doing* a job.

The underlying structure of this system is modelled as a class diagram in Fig. 1. The association between classes Worker and Car represent which workers share the same car. The association between classes Worker and Job represents which workers are *needed* for doing a job. A worker can be *assigned* many jobs – but a worker can only do one job at a time. Note that a single job may need more than one worker to participate.

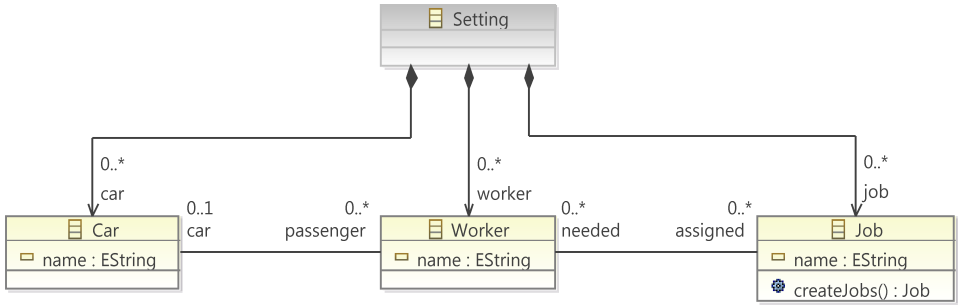

Fig. 1. Structural model

Figure 2 shows an object diagram with an example situation of a company with some workers, jobs and cars and how they are related.

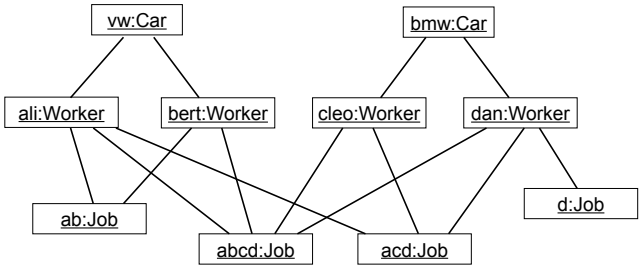

Fig. 2. Some configuration of a company

For modelling the system's behaviour, the ECNO distinguishes between the *life cycles* of objects and the *coordination* of the behaviour among the different objects. For clarity, objects that have a life cycle and for which the ECNO defines how to coordinate their behaviour are called *elements*.

The basis for defining and coordinating behaviour in ECNO are *events*. To this end, the ECNO allows us to define the types of events in the system. In our workers example, the events are arrive and depart, which mean that the workers and cars are arriving at or departing from the work site. Moreover, there is the event doJob, which means that a job is done; and there is an event cancelJob, which means that an existing job is cancelled.

These events are formally defined in the ECNO *coordination diagram* of Fig. 3 – shown as rounded rectangles. More importantly, the coordination diagram defines the coordination of how different elements are supposed to participate in events together. To this end, it equips some parts of the structural model from Fig. 1 with some additional annotations, which are explained below.

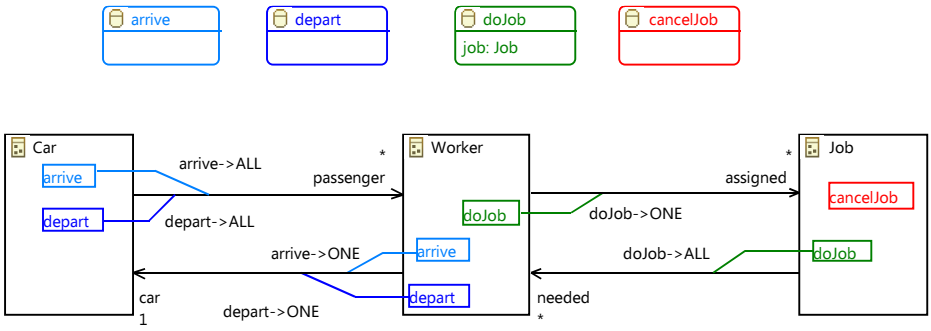

Fig. 3. Coordination diagram

When a Car is ready to participate in an arrive event, the coordination diagram requires that all workers sharing that car participate in this arrive event, too. To this end, there is a box with label arrive in the Car. This box is called a *coordination set* for event arrive of element Car. This coordination set is linked to the reference passenger with an annotation arrive->ALL. This annotation is called a *coordination annotation* and says that for a given car participating in an arrive event, every passenger (i. e. every worker at the other end of the link corresponding to passenger in the given situation) must also participate in the arrive event. Now, for a Worker participating in an arrive event, there is another coordination set for arrive, which imposes additional requirements of other elements participating. The Worker has a coordination set with a coordination annotation arrive->ONE linked to reference car. This means that for a Worker participating in an arrive event, there must be one Car at the end of the link car that must also participate in that arrive event. This gives us a combination of different elements participating in different events. Once such a combination of elements and events meets all the coordination requirements of the coordination diagram for each event, we call this combination an *interaction*. An interaction can be *executed*, which means that all involved elements execute the associated events in an atomic way.

Now, we assume that all workers have arrived at work and that the configuration is as shown in Fig. 2. This means that, according to their life cycle, each worker can participate in a doJob event. Let us assume that cleo would want to participate in a doJob event. Since there is a coordination set for event doJob for Worker, other elements would be required to participate. The coordination annotation doJob->ONE would require one of the jobs assigned to cleo to participate in the doJob event, too. In our example, there are two possibilities: job acd and abcd. Let us investigate job acd. The coordination set of Job for event doJob with the coordination annotation doJob->ALL linked to reference needed, specifies that all workers at the end of the link also need to participate in the doJob event. For the job acd, this would be the workers ali, cleo, and dan. The worker cleo is already participating in the doJob event – we started from there. But, now ali and dan also need to participate. This way, the coordination diagram makes sure that also all needed workers participate in the doJob event when the job acd does.

Note that, in our example, all elements participate in the same event. In general however, there can be more events involved in an interaction, and even a single element can participate in different events at the same time.

As mentioned before, the coordination diagram defines only the coordination of the behaviour among different elements. The life cycle for each element is defined separately: the life cycle defines when an element can participate in an event. In ECNO, one possibility for defining the life cycle for an element is through *ECNO nets*, which are a form of Petri nets. Figures 4 and 5 show the life cycles of a worker and a car, respectively. We omit the life cycle of the jobs here. The transitions labelled with the respective events indicate when the element can participate in which event.

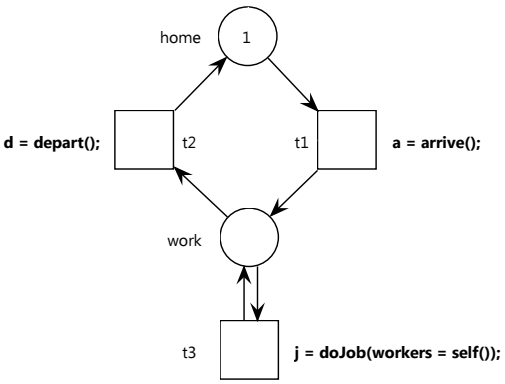

Fig. 4. Life cycle of a Worker

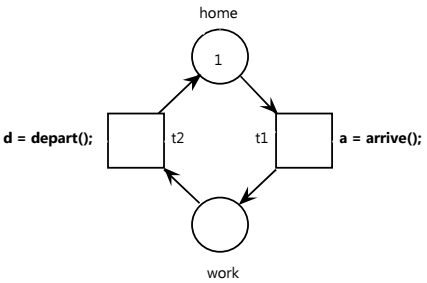

Fig. 5. Life cycle of a Car

In this paper, an ECNO model consists of a structural model, the coordination diagram and the ECNO nets for the life cycles of the elements.

3 ECNO State Space

The state space of such an ECNO model, basically, consists of the states, which represent situations like the one shown in Fig. 2, where additionally for each element the state of its life cycle of and the value of each of its attributes would be stored. In our case, the state of the life cycle would be a vector of natural numbers representing the marking of the resp. ECNO net.

The transitions of the state space would be the interactions. For each interaction in the state space, the involved elements, the links between them and which events are associated with which elements and links are represented.

Figure 6 shows an abstract representation of a part of the state space of the example from Sect. 2. The details of the states are not represented in that figure at all, since the focus of the validation is on the interactions.

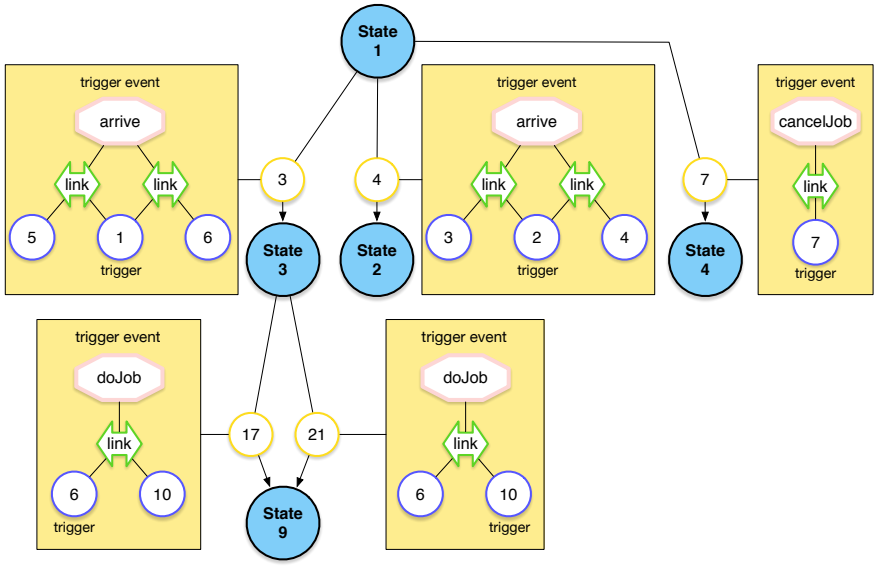

Fig. 6. Abstract representation of a part of the state space

The ECNO Tool[4] provides a *state space generator*, which systematically generates the state space for an ECNO application starting from some initial configuration. Actually, it is also possible for the user to run the ECNO application normally and record the fragment of the state space which the application comes by while running.

[4] See http://www2.compute.dtu.dk/ ekki/projects/ECNO/

4 Traces

As an abstraction of the behaviour of the implementation of the system, we use *traces*. Each trace corresponds to one execution of the system, which is represented as a sequence diagram showing how the different parts of the system invoke each other. Figure 7 shows an example of a trace of the final implementation of our example. This trace corresponds to the workers cleo and dan arriving at work together and then dan doing job d. The separation of the two steps is indicated by the green bars, which represent a state in the state space. But the green bars are actually not part of the trace. They will be computed later when mapping the trace to the state space.

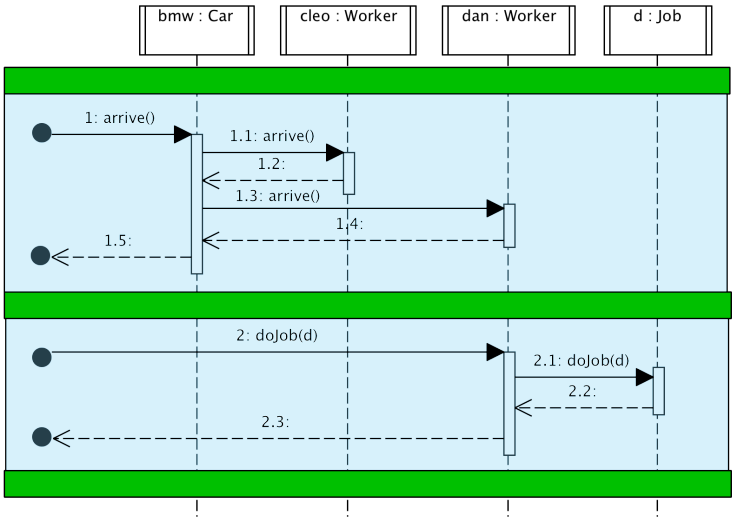

Fig. 7. Implementation behavior: Traces

Egilsson [5] implemented a tracer that could record a trace in the running implementation. It basically used AspectJ for recording the relevant method invocations.

5 Mapping Traces to State Space

The most important part of validating an implementation against an ECNO model (or actually its state space) is identifying parts of the trace which correspond to an interaction in the state space. These parts will be called segments of the trace (the parts between the green bars in Fig. 7). The tricky part is that interactions in ECNO are executed atomically, whereas the segments in traces are not executed atomically and can actually be intertwined. But, the segments

of the communication in the sequence diagram need to correspond to a complete interaction in the state space. And the validation consists of computing segments in traces and mapping them to interactions in the state space.

The mapping algorithm looks for the communication among the same elements in the trace, which also can be found as a link in an interaction. But, there can be more than one invocation for one link between these elements in a trace and the order can be arbitrary. The mapping algorithms starts matching the communication from the beginning of the trace to the interactions in the state space starting from its initial state. And once all communication for an interaction is found the segment is created at this point; the end of which corresponds to the next state. From there, the process continues with a lot of backtracking for alternative matches.

Egilsson [5] implemented such a mapping algorithm and showed that computing such mappings was feasible for the original banking example. Figure 8 shows the idea of this mapping with a short trace and a small excerpt of the state space from the original banking example.

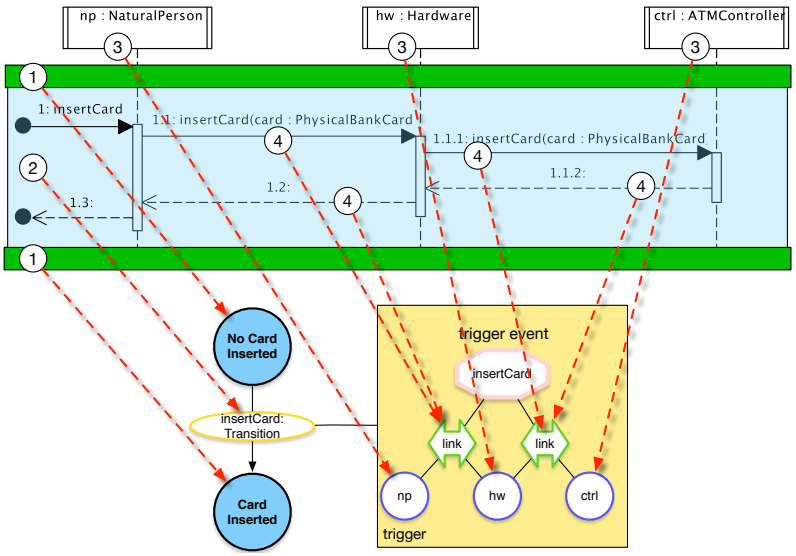

Fig. 8. Mapping

Of course, there are much more details in the mapping algorithm, which we cannot discuss here (see Egilsson [5] for more information).

6 Conclusion

In this paper, we have discussed the main idea of how to validate the final implementation of a system against its original ECNO specification. More details can be found in Egilsson's master thesis [5]. But even this thesis is just the beginning, since ECNO has more features which are not yet covered by the mapper implemented in the master thesis.

References

1. Kindler, E.: Coordinating interactions: The Event Coordination Notation. Technical Report DTU Compute Technical Report 2014-05, DTU Compute, Kgs. Lyngby, Denmark (2014)
2. Jepsen, J.: Realizing a workflow engine with the Event Coordination Notation. Master's thesis, Technical University of Denmark, DTU Compute (2013) IMM-M.Sc.-2013-101.
3. Jepsen, J., Kindler, E.: The Event Coordination Notation: Behaviour modelling beyond mickey mouse. In Roubtsova, E., McNeile, A., Kindler, E., Gerth, C., eds.: Behavior Modeling – Foundations and Applications. International Workshops, BM-FA 2009-2014, Revised Selected Papers. Volume 6368 of LNCS. Springer (2015)
4. The EU FP7 MIDAS project - 318786: Deliverable 5.1: Methods and tools for the intelligent planning and scheduling of test campaigns, http://web.itainnova.es/midas/dissemination/public-deliverables/d5-1-methods-and-tools-for-the-intelligent-planning-and-scheduling-of-test-campaigns/. (2013)
5. Egilsson, P.I.: Model-based software validation: Validating distributed systems against ECNO specifications. Master's thesis, Technical University of Denmark, DTU Compute (2016)